Key Aspects of Factory Design Pattern:

1. **Encapsulation of Object Creation:** The factory pattern hides the instantiation logic of classes and centralizes object creation, allowing the client to interact with an interface rather than specific implementations.
2. **Abstraction:** It defines an interface for creating objects and allows subclasses to decide which class to instantiate, promoting loose coupling.
3. **Return of Common Interface:** The factory returns an object that adheres to a common interface or abstract class, promoting polymorphism.
4. **Extensibility:** It allows for easy addition of new types of products without modifying the client code.

Advantages of Factory Design Pattern:

1. **Loose Coupling:** The client does not need to know the details of the concrete classes. It only interacts with an interface or abstract class, making the system loosely coupled.
2. **Single Responsibility Principle:** The object creation logic is encapsulated in the factory, making the code easier to manage and maintain.
3. **Scalability:** You can add new classes (products) without altering the existing factory or client code, making the pattern scalable.
4. **Promotes Reusability:** Since object creation logic is centralized, it can be reused across multiple client applications.

Disadvantages of Factory Design Pattern:

1. **Increased Complexity:** Introducing a factory adds more classes and abstraction layers, which may increase the complexity of the code, especially for simple applications.
2. **May Lead to Class Proliferation:** Every time a new product type is added, you need to create a new class and update the factory, leading to more classes in the system.
3. **Maintenance Overhead:** If the factory logic becomes too complex or involves many classes, it can become difficult to maintain and debug.
4. **Not Always Needed:** For small applications where object creation logic is straightforward, using a factory can be overkill.

When to Use the Factory Design Pattern:

- When the exact type of object to be created is determined at runtime.
- When you want to centralize object creation logic to promote loose coupling.
- When you need to instantiate classes from a family of related or dependent objects.